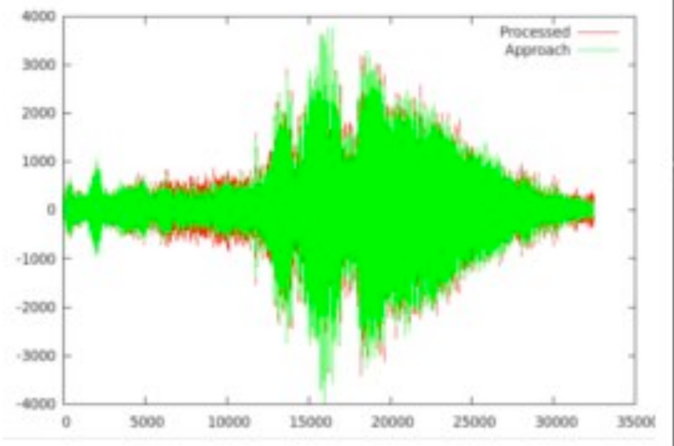
An Architecture Using a Finite Difference Method to Calculate Realistic Sound Equalization in Games

B. Moreira
E.W. C. Gonzales
M. Kischinhevsky
MediaLab
Computer Departament
Universidade Federal Fluminense

C.L.Kuryla*
Florida International University

D. Brandão**
Centro Federal de Ensino Tecnológico
Celso Suckow da Fonseca - CEFET/RJ
Computer Departament
Universidade Federal Fluminense



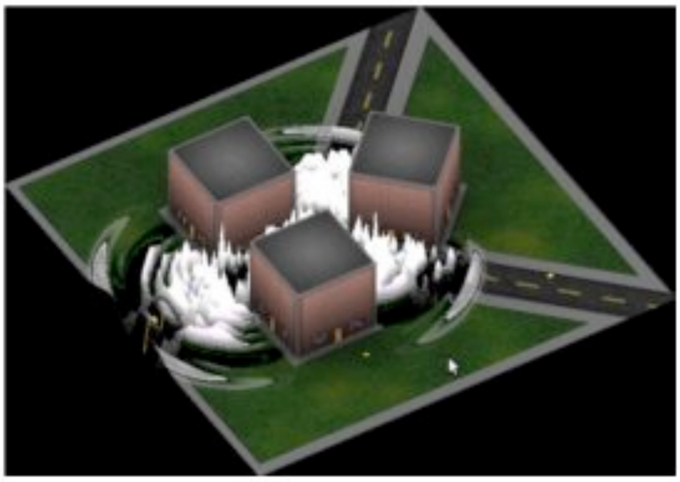


Figure 1: *Illustration of the proposed Sound propagation approach.*

Abstract

Most games and other interactive virtual environments focus on rendering natural phenomena in the most believable manner by using accurate visuals and physics. However, not much effort has been put into accounting for the physics of sound. The simulation of the real behavior of sound through an environment, when considering the speed of sound, reflection, and absorption, is computationally expensive and is usually left aside. In this work, an algorithm that calculates sound wave propagation using a finite difference method is used and extended to present a novel approach to sound rendering. This approach reaches the objective more quickly, and the sound generated has no perceptible loss of accuracy. The approach is designed to be implemented in GPU architectures and eventually enable real-time results.

Keywords:: Sound Wave Propagation, Finite Difference Method, GPGPU, Audio Effects

Author's Contact:

{bmoreira,esteban,kisch}@ic.uff.br
*ckury001@fiu.edu
**dbrandao@ic.uff.br

1 Introduction

Games are constantly being improved to describe reality in a more believable way. Many electronic games, especially those which are classified as simulation-games, rely on the modeling of natural phenomena in order to immerse the player in the game and promote the suspension of disbelief. However, simulating sound wave propagation has not been considered a priority, due in part to its computational complexity. Positional audio libraries have signifcantly contributed to this field, but the industry, as well as most academic research, has always been focused on providing more realistic graphics and, to some extent, rigid-body physics simulations. Positional audio libraries are now a common commodity and current implementations focus only on calculating intensity and pan based on the relative position and orientation of the sound source and the virtual listener. These libraries do not consider important issues such as the time the sound takes to travel through the environment (i.e. air), reflection, and the absorbency of the objects present in the scene. Taking scene geometry into account would greatly improve the immersion experience. Scene geometry affects the way sound travels through the virtual environment, which often changes the perceived equalization and the direction from which the listener hears the generated sound.

Despite the obvious benefits, simulation methods for such phenomena are computationally intensive, and running them in real-time, concurrently with all of the other expensive tasks, has not been a viable option. However, current Graphics Processing Unit (GPU) [GPGPU 2010] technology has the computing power to run and implement a simulation like this in real-time without compromising the overall performance, i.e. the GPU works as mathematics coprocessor.

When determining the scene geometries effect on sound traveling through an environment, it is necessary to use a mathematical model that represents this propagation physically. In order to solve this model computationally, two methods commonly used are finite difference and finite element methods. Finite element methods are more computationally expensive than finite difference methods with an explicit approach, so the choice of the second method is more appropriate for a game environment. [Zamith et al. 2010] implemented an algorithm that uses the finite difference method in a GPU. This work shows an accurate method for calculating the scattering of acoustic waves in a non-homogeneous medium and can be done in real-time.

In [Zamith et al. 2010], the sound wave propagation calculation was used to consider scene geometry while rendering visual effects and check the direction of the sound when it reached the listener. In this paper, the previous work is extended and a first approach is proposed using a finite difference method implemented in GPUs to do the actual rendering of a sound that travels through a medium and is heard by a listener at some distance.

After implementing code that took an audio file as input data for the source position, as in [Zamith et al. 2010], code was created to capture the audio data at the destination (listener position) and generate the output sound. When these functions were implemented, it was found that with a sample rate of 44 kHz, the method could reach a real-time processing speed for visual effects (as shown in Figure 1-right image), but not for sound rendering. This work proposes an approach to capture the output audio in a faster way by using a partial processing of the wave propagation, while still obtaining results which are perceptually acceptable and close to how a human would hear the sound if it were totally processed by the method.

2 Related Work

Techniques for simulating sound propagation may be classified into two categories: geometric acoustics (GA) and numerical acoustics (NA). Previous works based on numeric methods are in general too computationally expensive for real-time use, while approaches based on geometric techniques do not accurately represent the physics of sound because they overlook effects such as diffraction and scattering.

Geometric acoustics assumes rectilinear propagation of sound waves, so most approaches make use of ray-casting [Röber et al. 2007] and beam tracing [Funkhouser et al. 2004; Antonacci et al. 2004] techniques.

Numerical acoustics directly solves the wave equation governing physical propagation of sound in a scene. With enough computation, all wave phenomena can be captured, including diffraction and scattering in complex scenes. Numerical approaches are insensitive to the complexity and polygon count of a scene and instead scale with its physical dimensions. Two methods commonly used are finite difference and finite element methods. Several works demonstrate the use of these methods, such as [Balevic et al. 2008; Michea and Komatitsch 2010; Raghuvanshi and Nico 2008; Savioja et al. 1994; Kowalczyk and Walstijn 2008], but they are not suitable for real-time use.

Sound is considered to have low priority when making games realistic, so few works attempting to realistically render sound in games were found. [Raghuvanshi et al. 2010; Siltanen 2010] uses a technique with a pre-computed scenario, but a problem arises when considering memory usage and the changing state of the game at runtime. Moreover, all the GPU and CPU resources are typically used for physics and computer graphics calculation. Since CUDA architecture basically allows only one concurrent thread, most engines and commercial applications prefer to use the computer horsepower for these tasks, so not many resources are available for the sound computation. However, new architectures of GPUs are allowing the execution of concurrent kernels, which allows real-time sound rendering to be done in parallel with the computer graphics and typical physics calculations. No previous work was found that uses numeric methods without pre-computed scenarios to take the scene geometry into account and still reach a real-time sound rendering that is close to reality.

3 Acoustic Wave Equation

In this section, the mathematical model that physically represents the scattering of acoustic waves and the numerical approach adopted are presented. The wave equation is a second order differential equation that describes the behavior of sound waves over time. The acoustic wave field is described by $P(x, y, t)$ and $u(x, y, t)$, where P is the pressure field and u is the particles displacement.

The relation between P and u is given by $P(x, y, t) = -k\nabla^2 u(x, y, t)$. Thus, the 2D wave equation can be represented by Equation 1;

$$\frac{\partial^2 P(x,y,t)}{\partial t^2} = c^2 + \left[\frac{\partial^2 P(x,y,t)}{\partial x^2} + \frac{\partial^2 P(x,y,t)}{\partial y^2}\right] + f(x,y,t) \tag{1}$$

where x and y are Cartesian coordinates, t is time, $c = c(x, y)$ is the velocity of an acoustic wave, and $f(x, y, t)$is the source term.

To numerically solve the partial differential equation (PDE), several techniques may be employed. The method that was chosen is based on the Finite Difference Method (FDM), which replaces the quest for a solution on the continuous domain by one on a finite number of points, the grid points, which cover the whole domain. Specifically, after the discretization of the PDE, one obtains a set of equations, the finite-difference (FD) equations, which approximate the differential equation at all grid points. Each FD equation around a grid point is written as an algebraic expression which involves its neighboring points to replace the spatial derivatives.

The approximation of the PDE at the grid points is done through spatially centered differences, which offer second order spatial accuracy. The technique used for the time discretization in this work is second order as well. The proper algebraic expressions can be obtained with Taylor series expansions. Using a second order approximation for space and time, as explained, and assuming that $h = \Delta x = \Delta y$ and $t = n\Delta t$, the wave equation is rewritten in its discretized form as Equation 2;

$$\begin{aligned} P_{i,j}^{n+1} = 2P_{i,j}^{n} - P_{i,j}^{n-1} + \Delta t f(x,y,t) + \\ \frac{A}{12}\left[P_{i-1,j}^{n} + P_{i+1,j}^{n} - 4P_{i,j}^{n} + P_{i,j-1}^{n} + P_{i,j+1}^{n}\right] \end{aligned} \tag{2}$$

where $A = \left(\frac{c(x,y)\Delta t}{h}\right)^2$ and $n = 1, 2, 3, ...$ represents the time instant. Since the velocity field does not vary with time, it is not a function of time [Golub and Ortega 1991].

The discrete expression in Equation 2 provides explicit computation of approximate values at time instant $(n + 1)$, once those at time instants $(n - 1)$ and (n) are available at all grid points. The explicitness is a convenient feature that allows efficient use of parallelism since each grid points value at the $(n + 1)$th time instant is computed independently. Another aspect of discrete approximations for differential equations is that some implicit discretization strategies provide unconditional stability, while the one employed here provides conditional stability. That is, time steps have to be less than an upper limit, above which the numerical solution blows up (the upper limit can be found with the von Neumann criterion), i.e. its amplitude increases non-physically. On the other hand, unconditional stability would allow larger time steps than the explicit methods upper limit. However, the implicit method is not efficiently parallelized on GPUs. Consequently the explicit technique of Equation 2 was chosen, and a careful choice of time steps was made.

4 Finite Difference Method on GPU and Implementation Aspects

The Graphics Processing Unit (GPU) is a hardware specially designed to handle tasks related to visual effects. A GPU can be considered similar to a CPU, but with different processing units that are designed specifically for graphical data processing. Modern high-end GPUs have computational power far greater than CPUs.

Before CUDA, the concept of GPU Computing was to map the problem as a set of vertex and fragments to generate a texture representing the final desired solution. Since 2006, with the NVIDIA CUDA release[Nvidia 2010], the world of high-performance computing became more accessible. A GPU with hundreds of cores is now available for a tenth of the price of a cluster with the same computational power.

CUDA extends existing programming languages by adding a set of instructions that allow code execution on NVIDIA GPU's. As a consequence of its original design, which was for visual effects, GPU memory structure is divided into global, texture, and shared memories. Shared memory is a small but extremely fast memory. This speed comes from its physical proximity to the processing core. Texture memory is a read only memory which is slower than

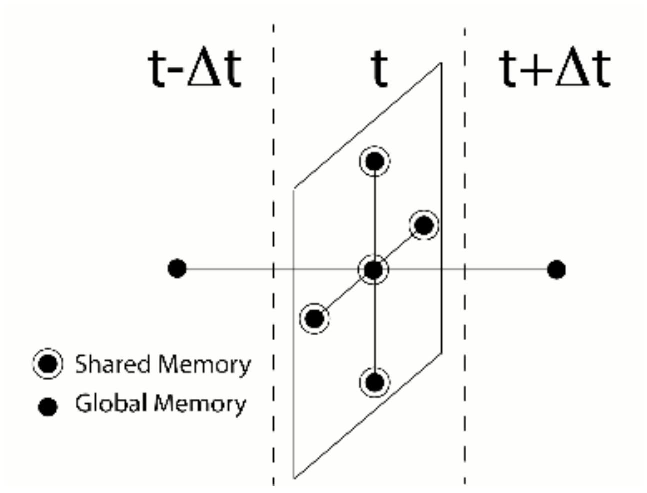


Figure 2: *Second order 2D finite difference operator. To compute the next step $(t + \Delta t)$ it is necessary to fetch the actual (t) and the past one $(t - \Delta t)$.*

the shared memory, but is almost twice as fast as the global memory.

As described in [Brandao et al. 2010; Sabino et al. 2011], efficient finite-difference implementations take advantage of shared memory. Using a single GPU, the amount of memory that needs to be allocated for the iterative solution of Equation 2 is equal to three times the domain size. The past and present time instants and the velocity field need to be available at any time. Figure 2 shows a 2D FD explicit scheme with a second order spatial FD operator.

A typical CUDA application workflow consists of four basic steps: (1) initialize the necessary data on the host (CPU side), (2) copy the data from the host to the device (GPU side), (3) invoke the kernel that will process the data in the device, and (4) read the data back to the host.

For the FD problem, the velocity field is initially allocated and sent to the device memory. Sufficient memory is initialized and allocated to hold values for past and present instants of time. The amplitude values for $P^0(x, y)$ and $P^1(x, y)$ are determined using a forward explicit approximation FD scheme. Equation 2 is then used for subsequent time steps.

A GPU can run millions of lightweight threads efficiently. To help with the management of these threads, CUDA uses the concept of a grid of thread blocks. Threads are organized into blocks, which in turn are organized in a grid. The proposed approach computes each new value $P^{(n+1)}(x, y)$ with only one thread. To compute the next step $(t + \Delta t)$ it is necessary to fetch the actual (t) and the previous one $(t - \Delta t)$.

For efficient memory access, shared memory is used to hold values of some value in a simulation step. This avoids unnecessary reads from global memory. Figure 2 shows that a thread must access up to five values of the same instant $P^{(n)}(x, y)$. Bringing these values into shared memory makes the accesses much faster. The values of $P^{(n-1)}(x, y)$ are fetched directly from global memory. Since all the threads of the same block will only access one position, a coalesced memory read taking maximum advantage of the performance of GPU architecture can be guaranteed.

5 Real Audio Reproduction

This section explains how an audio file works and how the implemented method processes it to generate what this work considers to be the real behavior of sound or the sound totally processed. This will provide a basis for the next section, which will explain the heuristic used for a faster calculation.

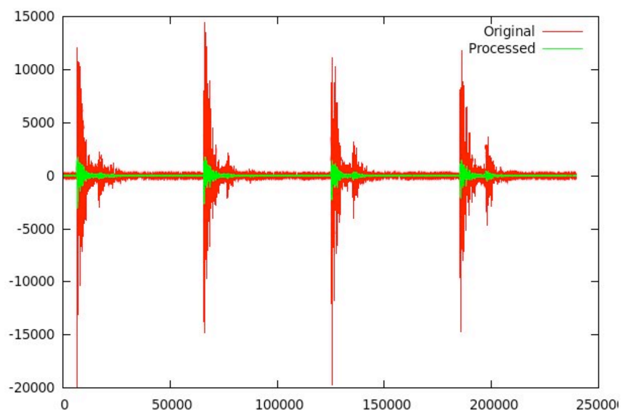


Figure 3: *The comparison between the original sound file footsteps.wav and the processed file. The x-axis represents the sample number and the y-axis is the amplitude.*

5.1 Audio file

An audio file is composed of sample values that represent the discretization of the wavelength passing through the point of capture at a given time interval [Farnell 2008]. This file has two parameters: the number of samples per second, and the accuracy of the sample. The former indicates how many wavelength values are analyzed per second. The higher the value of samples per second, the greater the number of frequencies the audio file can represent.

According to sampling theory, the number of sample points per second must be at least two times greater than the highest frequency in a signal. In order to cover the human hearing range, from 0Hz to 20kHz, it is necessary to have at least 40000 sample points every second [Farnell 2008]. The second parameter describes the order of approximation. It is the precision of a sample (resolution or sample size), i.e. the number of bits that are reserved to represent a single sample value (16,32,64-bits, etc.). The higher the value per sample, the more accurate the audio will be. However, the accuracy also depends on the audio input and output hardware, so a very high precision value for each sample time is usually not required.

5.2 Audio processing

To perform the audio processing using finite differences, pulses that correspond to the samples in a sound file are inserted at a point in the environment. This point is designated as a source (representing a radio or speaker, for example), so that at each iteration the method will calculate the propagation of the pulses that come out from the point. The pulses are given until the audio file ends, at which point, the source stops emitting sounds.

For each iteration of the finite difference method, the amplitude of the waves propagated through the whole domain for a chosen time interval can be calculated. Then, a sample of sound arriving at a specific place can be read, using the considered method, by reading the amplitude value at the specific place for a time instant. To calculate the real behavior of a sound at a point of interest, the samples at the point are read using the time interval required for the given/desired precision of audio frequency, for example, 40000 samples per second. Reading the samples during the time of interest will then give the real sound as it would be heard at the position where the read was done. The sample values are stored and then used to generate a new sound file. This new file contains the sound observed when considering the real propagation of the waves through the scene.

In Figure 3 an original and processed audio file are shown together. This figure experiment used a sound source at position $(x, y) = (64, 20)$ in an open scene with the listener positioned at $(x, y) = (64, 64)$, simulating a distance around 44 meters.

The above figure confirms the intuition that when the receiver and source are separated by a sufficient distance, the sound amplitudes are attenuated because of the dispersion of waves through the environment. The figure also demonstrates that the amplitude variations of the processed and original sounds are very similar, which means that when the new sound is played, it closely resembles the original sound, but with a lower volume.

The problem with this real reproduction of sound is the amount of computing time required. It is not fast enough for a real-time algorithm, and as the aim here is for use in games, it must happen in real-time to be considered plausible for use. The results in the following sections show the average processing time for different audio files under the conditions described for Figure 3. The times are clearly not viable for use in real-time applications.

The objective is now to create a method that not only considerably reduces the processing time for the rendering of audio files, but also maintains sufficient audio quality, so that the quality loss when using the method is imperceptible to humans. This way, the resultant audio file will sound realistic (or very close) when heard. The proposed method in this work is still not suitable for real-time use. However, it opens new possibilities for studies and improvements in the algorithm, possibly enabling future work to attain a processing time suitable for the purposes and requirements of games.

6 The Proposed Approach

The implementation of the method that considers the wave propagation through the scene, and the subsequent analysis of the resulting sounds after they were completely processed, revealed the necessity for the development of a new approach which could still make use of the behavior of the propagation of a real wave, but possibly reproduce the behavior for the whole audio file. This could result in a significant reduction in the audio rendering time.

To address this improvement, a new approach was developed. A method was created to analyze only one pulse of a wave, and then reproduce the pulse behavior for an entire audio file. The pulse is emitted from the source position and analyzed over an interval of t seconds. In this work, the time interval used is 200ms, represented by 8000 samples considering a system sample rate of 40kHz. This interval was chosen only as an interval for first results; the criteria taken into consideration to select this time were only that it was not too short, so it was not short enough to lose some effects like late reverberation and echo, and that it was not too long, so it was not long enough to interfere with the calculation of the behavior of the wave by sampling at a time after the wave had already spread out and attenuated.

This single pulse analysis is done by reading the amplitude values that reach the point (x,y) where the listener is positioned. This is done in the same way as it is done for the entire audio processing as described in Section 5.2. An example of the behavior of a single propagated pulse is shown in Figure 4. After the single pulse propagation is calculated, the values are compared to the original pulse value, and the comparison yields the proportional values based on the propagation time and the original amplitude.

After the analysis step comes the reconstruction phase. In the reconstruction phase, the system scans the whole original audio file, and for each sample, applies the values that would represent the same behavior as calculated through the pulse analysis. These values (that represent the behavior of each sample) are stored in memory and used to generate the new sound. Each sample generates a window of values representing its individual behavior; this window will affect the next 200ms of sound (this was the chosen time of pulse observation). As shown below, this behavior will overlap for each sample, and when this happens the values are added. This summation can be done because of the superposition property of acoustic waves[Farnell 2008]. Figure 5 shows how the heuristic works.

7 Results and Conclusion

Our implementation was made in C++ using OpenGL for visualization and NVIDIA CUDA API for GPU processing. We used irrKlang to manipulate audio files for reading and playing. The computational environment employed was a TESLA C1060 composed of 30 multiprocessors, and a quad core CPU with 2.4GHz, 4GB DDR3 memory, running on a Linux OS.

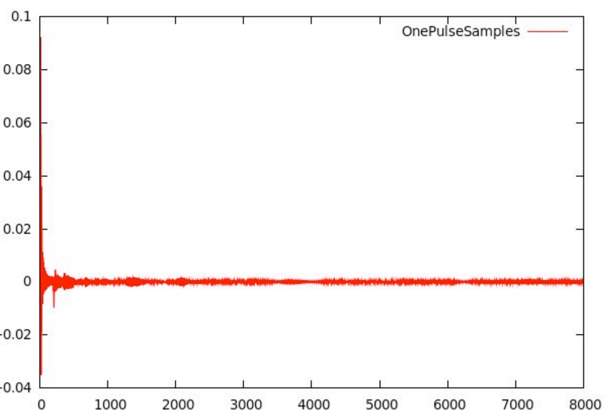


Figure 4: *The behavior of a single pulse propagation analyzed for 200ms.*

For the comparative tests (see Table 1) the scene used was constructed as follows: the sound source was at position (x,y) = (64,20) in an open environment, with the listener established at (x,y) = (64,64), simulating a distance of 44 meters from one to the other. Four different audio files were used to compare the processing time for different quantities of samples per file. As shown in Table 1, significantly less time is needed to reproduce the sound when using the approach presented in this work than the time needed to totally process the audio for the real behavior.

Table 1: *Comparative results of different audio files with different numbers of samples in a* 128×128 *grid. The results are in seconds and represent the time taken to render the complete audio or to render the audio using this work's heuristics. Errors between the methods are shown in last two columns.*

File Name Approx Total Time	Number of Samples	Time to render: real behavior of sound file(s)	Time to render: pulse analysis heuristics (s)	Error	Error (db)
bell.wav / 00:01	32449	84,685	23,295	116	1.99
baby.wav / 00:01	169984	444,234	39,382	45	1.07
footsteeps .wav / 00:02	239616	609,182	47,258	11	1.61
crowdtalk .wav / 01:00	5765766	14702,703 (estimated - not rendered)	710,932	N/A	N/A

As described in Section 6, the proposed method has two steps: analysis and reconstruction. For all of the files, the analysis step took 19 seconds (because the listener and source were in the same position in every trial), and the rest of the time was the reconstruction phase. To obtain a faster result, one suggestion is to have a preprocessing step to minimize (or even almost eliminate) the time required for the analysis phase. Another suggestion is to use a GPU calculation kernel that could solve the reconstruction step operations in parallel.

In reference to the sound quality, the test results were very encouraging. Figure 7 (at the end of the paper) shows a comparison between the original footstep audio file and the two methods used to render the processed sound: the real behavior obtained by the total processing and the behavior rendered by the proposed method. Observation shows the difficulty to even discern the difference between the two methods used to generate the processed sound since the results are so close. Figure 8 (also at the end) only shows the two methods, without the original file, for an easier comparison. Two additional audio file tests are shown in Figure 9 and Figure 10 (also at the end of the paper).

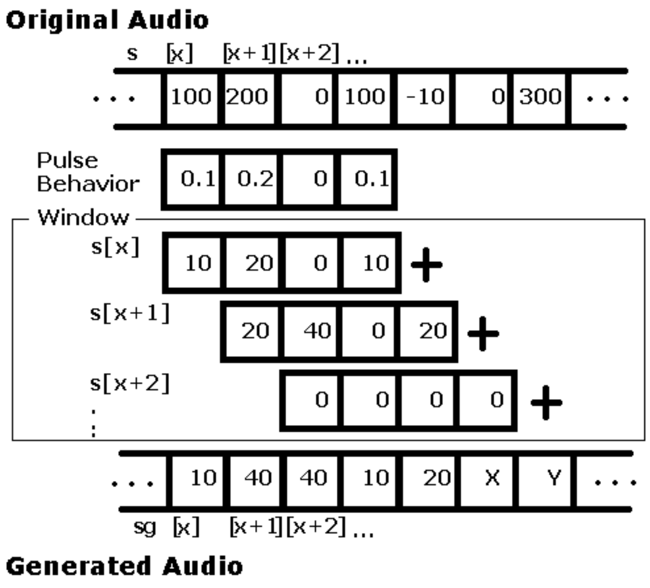


Figure 5: *Visual representation of the heuristic, using an example of a pulse with 4 samples shown.*

Observation of the superimposed graphs clearly shows that the results were very close, but the graphs are only part of the complete analysis. An error analysis was performed on the values to determine how close the proposed method came to the real, totally-processed result, as well as the perceptibility of the difference. The calculated errors are shown in the last two columns of Table 1. The errors in the amplitude values were first calculated using Equation 3:

$$Error = \frac{1}{m} * \sum_{1}^{m} (u^p - u^n) \tag{3}$$

where m is the number of audio samples, u^p is the sample value in the totally processed method, and u^n is the sample value the new proposed method. Equation 3 gives the error in the form of amplitude values, so further analysis regarding the perceptibility of the difference was possible. The relationship between amplitude and decibel is given by Equation 4:

$$dB = 20 * log_{10}\left(\frac{P}{P_{\max}}\right), \tag{4}$$

where $P_{\max}$ is the reference amplitude and P is the amplitude being compared [Farnell 2008]. Using Equation 4, the values in the last column of Table 1 were generated. The threshold for human loudness discrimination is roughly 1 dB over all audible frequencies [Jesteadt et al. 1977], so the differences between the results obtained from the new method and the real behavior method were almost imperceptible. Considering these tests, the proposed method provides good results in practice, and if this range of error persists for a large database, then sound rendered this way could be suitable for a game.

Additional tests were performed in an open scene, but with the addition of buildings, to examine the results obtained when other acoustic effects are present due to a different geometry. Effects such as reflections and diffractions can be observed in this type of simulation. Figure 6 shows one of the tests as an example. This test is not expressed in Table 1, but the figure shows that the rendering still produces a close result when using the proposed method.

As suggestions for future works, the authors would be interested in an improvement in the wave propagation method used, which in this work was the [Zamith et al. 2010] method, because [Zamith et al. 2010] uses a simple finite difference method of wave propagation that, for example, does not consider attenuation and absorption. Another suggestion for future work is to include an adaptive algorithm to calculate the number of samples needed for the pulse

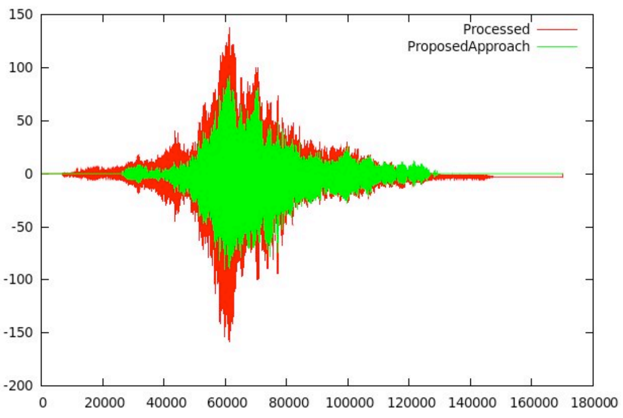


Figure 6: *Comparative graphic of the baby.wav file. The red is the audio as it reaches the listener in the position (x,y) = (84,20) after total processing, considering the source to be at (x,y) = (64,64). The green is the process done by the proposed method. In this test the environment had buildings and the listener was positioned behind a building relative to the source position used.*

in the analysis step. Using as few samples as possible without affecting the result would greatly improve the processing time for the reconstruction phase.

Acknowledgements

The authors gratefully acknowledge CNPq, CAPES and FAPERJ for the financial support of this work.

References

ANTONACCI, F., FOCO, M., SARTI, A., AND TUBARO, S. 2004. Real time modeling of acoustic propagation in complex environments. In *Proceedings of 7th International Conference on Digital Audio Effects*, 274–279.

BALEVIC, A., ROCKSTROH, L., TAUSENDFREUND, A., PATZELT, S., GOCH, G., AND SIMON, S. 2008. Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose gpus. In *IEEE International Conference on Computational Science and Engineering*, vol. 0, IEE, 327–334.

BRANDAO, D., ZAMITH, M., CLUA, E., MONTENEGRO, A., BULCAO, A., MADEIRA, D., KISCHINHEVSKY, M., AND LEAL-TOLEDO, R. 2010. Perfomance evaluation of optimized implementations of finite-difference method fow wave propagation problems on gpu architecture. In *Proceedings of the Computer Architecture and High Performance Computing Workshops*, Sociedade Brasileira de Computação.

FARNELL, A. 2008. *Designing Sound*. Applied Scientific Press, London.

FUNKHOUSER, T., TSINGOS, N., ARLBOM, I., ELKO, G., SONDHI, M., WEST, J., PINGALI, G., MIN, P., AND NGAN, A. 2004. A beam tracing method for interactive architectural acoustics. *The Journal of the acoustical society of America 115*, 2, 739–756.

GOLUB, G., AND ORTEGA, J. 1991. *Scientific Computing and Differential Equations: an Introduction to Numerical Methods*, 1st ed. Academic Press.

GPGPU, 2010. General-purpose computation using graphics hardware. `www.gpgpu.org`, February.

JESTEADT, W., WIER, C., AND GREEN, D. 1977. Intensity discrimination as a function of frequency and sensation level. *The Journal of the acoustical society of America 61*, 1, 169–177.

KOWALCZYK, K., AND WALSTIJN, M. V. 2008. Virtual room acoustics using finite difference methods. In *Proceedings IEEE Int. Symp. Communication, Control Signal Processing (ISCCSP)*, 1504.

MICHEA, D., AND KOMATITSCH, D. 2010. Accelerating a three-dimensional finite-difference wave propagation code using gpu graphics cards. *Geophysical Journal International 182*, 389–402.

NVIDIA. 2010. *Cuda Programming Guide*. Nvidia.

RAGHUVANSHI, N., AND NICO, G. 2008. Accelerated wave-based acoustics simulation. In *Proceedings of the 2008 ACM Symposium on Solid an Physical Modeling*, 91–102.

RAGHUVANSHI, N., SNYDER, J., MEHRA, R., LIN, M., AND GOVINDARAJU, N. 2010. Precomputed wave simulation for real-time sound propagation of dynamic sources in complex scenes. *ACM Transactions on Graphics 29*, 4 (July), 11.

RÖBER, N., KAMINSKI, U., AND MASUCH, M. 2007. Ray acoustics using computer graphics technology. In *Proceedings of the 10th International Conference on Digital Audio Effects*, 01–08.

SABINO, T., ZAMITH, M., BRANDAO, D., MONTENEGRO, A., KISCHINHEVSKY, M., LEAL-TOLEDO, R., SILVEIRA, O., BULCAO, A., AND CLUA, E. 2011. Scalable simulation of 3d wave propagation in semi-infinite domains using the finite difference method on a gpu based cluster. In *Proceedings of the V e-Science Workshop*, vol. 1, 110–118.

SAVIOJA, L., RINNE, AND TAKALA, T. 1994. Simulation of room acoustics with a 3-d finite difference mesh. In *Proceedings of Internation Computer Music Conference (ICMC94)*, 463–466.

SILTANEN, S. 2010. *Efficient physics-based room-acoustics modeling and auralization.* Master's thesis, Aalto University School of Science and Technology, Espoo, Finland.

ZAMITH, M., PASSO, E., BRANDAO, D., CLUA, E., MONTENEGRO, A., KISCHINHEVSKY, M., AND LEAL-TOLEDO, R. 2010. Sound wave propagation applied in games. In *Proceeding of IX Brazilian Symposium of Games and Digital Entertainment.*, Sociedade Brasileira de Computação.

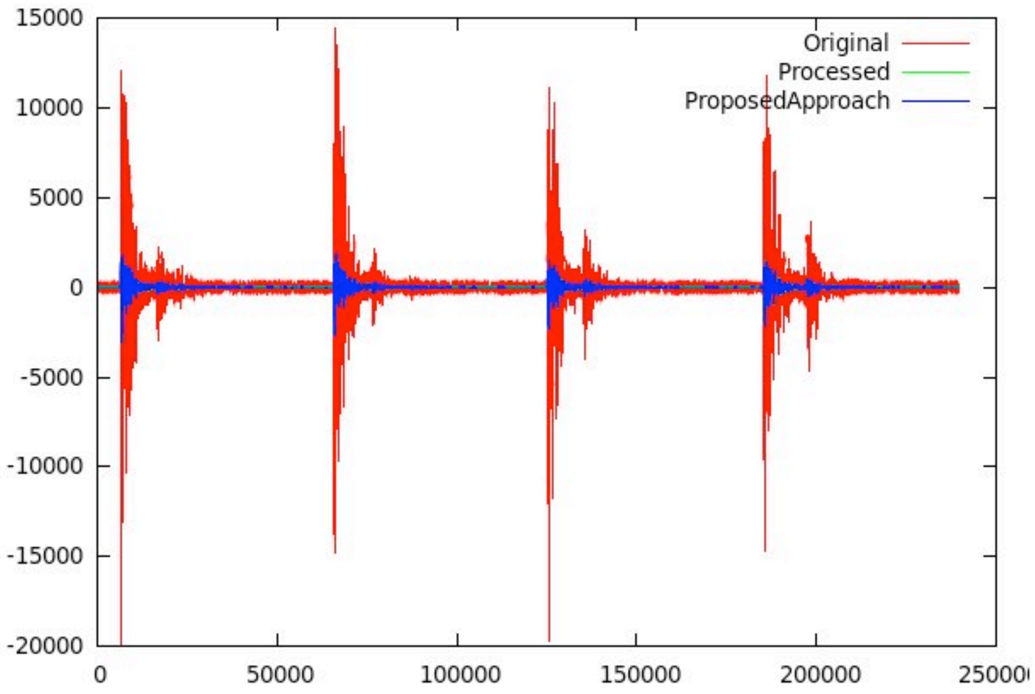


Figure 7: *Comparative graphic of the footsteps.wav file. Red represents the original sound. Green represents the real behavior audio arriving for a listener at (x,y) = (64,64). Blue represents how the sound arrives with the proposed method.*

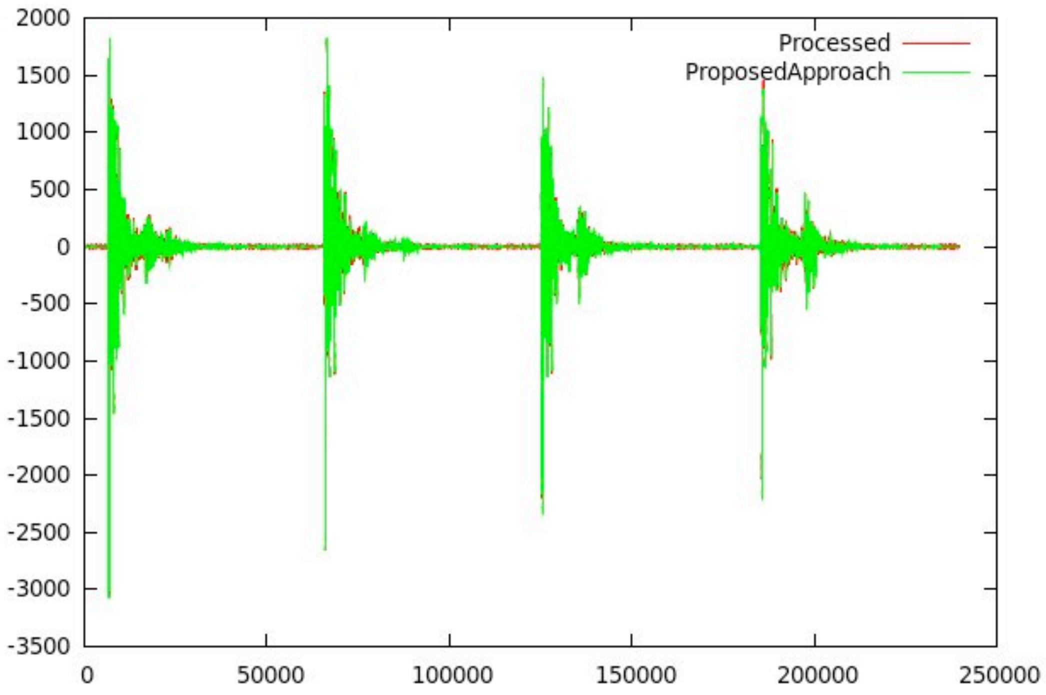


Figure 8: *Comparative graphic of the footsteps.wav file. Red represents the real behavior audio arriving for a listener at (x,y) = (64,64) after processing. Green represents how the sound arrives with the proposed method.*

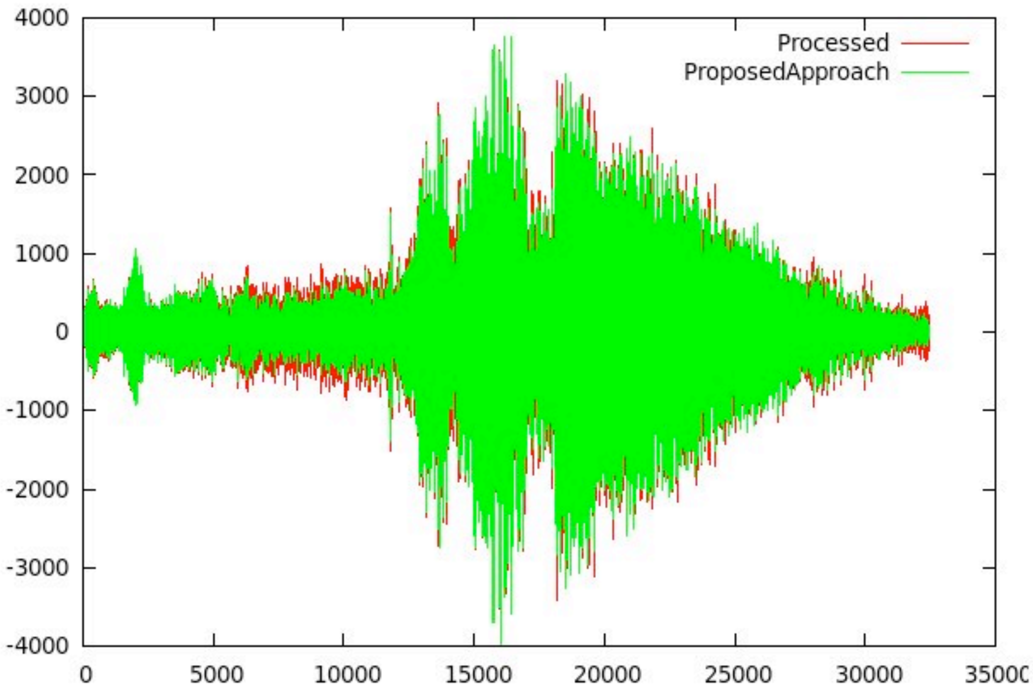


Figure 9: *Comparative graphic of the bell.wav file. Red represents the real behavior audio arriving for a listener at (x,y) = (64,64) after processing. Green represents how the sound arrives with the proposed method.*

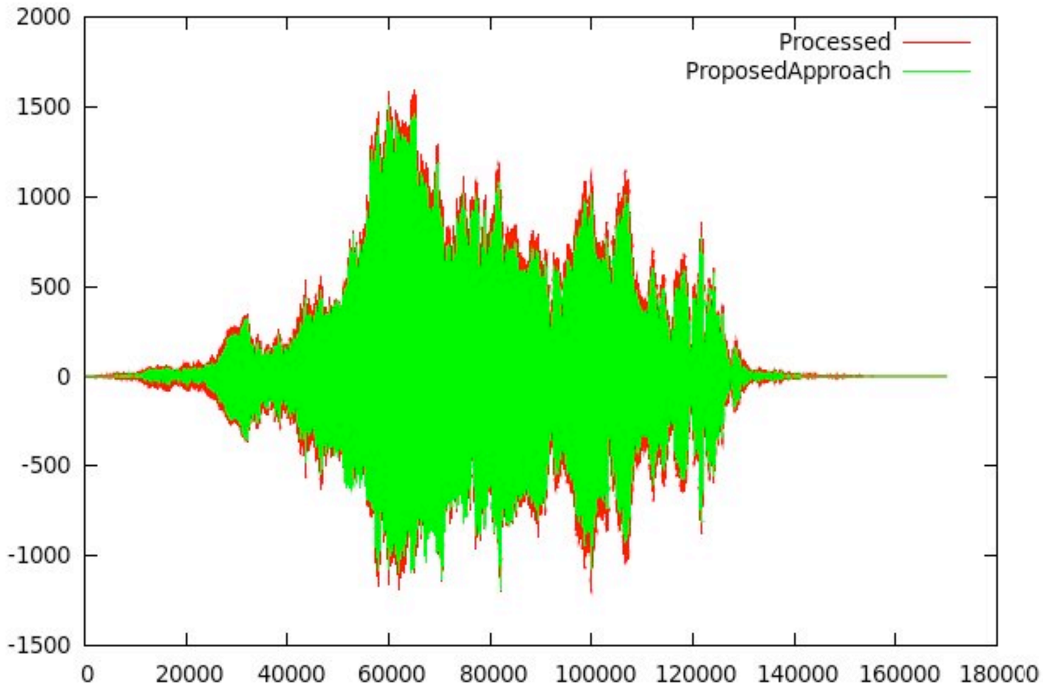


Figure 10: *Comparative graphic of the baby.wav file. Red represents the real behavior audio arriving for a listener at (x,y) = (64,64) after processing. Green represents how the sound arrives with the proposed method.*